Hadoop Streaming入门

Sep 28, 2015

hadoop (http://icejoywoo.github.io/category/#hadoop) hadoop (http://icejoywoo.github.io/tags/#hadoop) streaming (http://icejoywoo.github.io/tags/#streaming)

说明：本文使用的Hadoop版本是2.6.0，示例语言用Python。

概述

Hadoop Streaming是Hadoop提供的一种编程工具，提供了一种非常灵活的编程接口， 允许用户使用任何语言编写MapReduce作业，是一种常用的非Java API编写MapReduce的工具。

调用Streaming的命令如下（hadoop-streaming-x.x.jar不同版本的位置不同）：

```
$ ${HADOOP_HOME}/bin/hadoop jar ${HADOOP_HOME}/share/hadoop/tools/lib/hadoop-streaming-2.6.0.jar \
    -input <输入目录> \ # 可以指定多个输入路径，例如：-input '/user/foo/dir1' -input '/user/foo/dir2'
    -inputformat <输入格式 JavaClassName> \
    -output <输出目录> \
    -outputformat <输出格式 JavaClassName> \
    -mapper <mapper executable or JavaClassName> \
    -reducer <reducer executable or JavaClassName> \
    -combiner <combiner executable or JavaClassName> \
    -partitioner <JavaClassName> \
    -cmdenv <name=value> \ # 可以传递环境变量，可以当作参数传入到任务中，可以配置多个
    -file <依赖的文件> \ # 配置文件，字典等依赖
    -D <name=value> \ # 作业的属性配置
```

注意：-file是一个deprecated的配置，可以使用-files。

常见的作业属性

属性	新名称	含义	备注
mapred.job.name	mapreduce.job.name	作业名称	
mapred.map.tasks	mapreduce.job.maps	每个Job运行map task的数量	map启动的个数无法被完全控制
mapred.reduce.tasks	mapreduce.job.reduces	每个Job运行reduce task的数量	
mapred.job.priority	mapreduce.job.priority	作业优先级	VERY_LOW,LOW,NORMAL,HIGH,VERY_HIGH
stream.map.input.field.separator		Map输入数据的分隔符	默认是\t
stream.reduce.input.field.separator		Reduce输入数据的分隔符	默认是\t
stream.map.output.field.separator		Map输出数据的分隔符	默认是\t
stream.reduce.output.field.separator		Reduce输出数据的分隔符	
stream.num.map.output.key.fields		Map task输出record中key所占的个数	
stream.num.reduce.output.key.fields		Reduce task输出record中key所占的个数	

注意：2.6.0的Streaming文档中只提到了stream.num.reduce.output.fields， 没提到stream.num.reduce.output.key.fields，后续需要看下二者的关系。

stream开头的是streaming特有的，mapred.map.tasks和mapred.reduce.tasks是通用的

基本原理

Hadoop Streaming要求用户编写的Mapper/Reducer从标准输入（stdin）中读取数据，将结果写入到标准输出（stdout）中， 这非常类似于Linux的管道机制。

正因此，我们在linux本地方便对Streaming的MapReduce进行测试

```
$ cat <input_file> | <mapper executable> | sort | <reducer executable>

# python的streaming示例
$ cat <input_file> | python mapper.py | sort | python reducer.py
```

WordCount示例

准备数据

自行替换其中的<username>

```


$ cat input/input_0.txt
Hadoop is the Elephant King!
A yellow and elegant thing.
He never forgets
Useful data, or lets
An extraneous element cling!

$ cat input/input_1.txt
A wonderful king is Hadoop.
The elephant plays well with Sqoop.
But what helps him to thrive
Are Impala, and Hive,
And HDFS in the group.

$ cat input/input_2.txt
Hadoop is an elegant fellow.
An elephant gentle and mellow.
He never gets mad,
Or does anything bad,
Because, at his core, he is yellow.

$ ${HADOOP_HOME}/bin/hadoop fs -mkdir -p /user/<username>/wordcount

$ ${HADOOP_HOME}/bin/hadoop fs -put input/ /user/<username>/wordcount
```

编写Mapper

```
#!/bin/env python
# encoding: utf-8

import re
import sys


seperator_pattern = re.compile(r'[^a-zA-Z0-9]+')

for line in sys.stdin:
    for word in seperator_pattern.split(line):
        if word:
            print '%s\t%d' % (word.lower(), 1)
```

编写Reducer

```
#!/bin/env python
# encoding: utf-8

import sys

last_key = None
last_sum = 0

for line in sys.stdin:
    key, value = line.rstrip('\n').split('\t')
    if last_key is None:
        last_key = key
        last_sum = int(value)
    elif last_key == key:
        last_sum += int(value)
    else:
        print '%s\t%d' % (last_key, last_sum)
        last_sum = int(value)
        last_key = key

if last_key:
    print '%s\t%d' % (last_key, last_sum)
```

使用itertools.groupby的Reducer

```
#!/bin/env python
# encoding: utf-8

import itertools
import sys


stdin_generator = (line for line in sys.stdin if line)

for key, values in itertools.groupby(stdin_generator, key=lambda x: x.split('\t')[0]):
    value_sum = sum((int(i.split('\t')[1]) for i in values))
    print '%s\t%d' % (key, value_sum)
```

示例代码太过简单，应该包含更多的异常处理，否则会导致程序异常退出的。

调试方法

本地测试

前面说过，Streaming的基本过程与linux管道类似，所以可以在本地先进行简单的测试。 这里的测试只能测试程序的逻辑基本符合预期，作业的属性设置

```
$ cat input/* | python mapper.py  | sort | python reducer.py
a       2
an      3
and     4
anything        1
are     1
at      1
bad     1
because 1
but     1
cling   1
core    1
data    1
does    1
elegant 2
element 1
elephant        3
extraneous      1
fellow  1
forgets 1
gentle  1
gets    1
group   1
hadoop  3
hdfs    1
he      3
helps   1
him     1
his     1
hive    1
impala  1
in      1
is      4
king    2
lets    1
mad     1
mellow  1
never   2
or      2
plays   1
sqoop   1
the     3
thing   1
thrive  1
to      1
useful  1
well    1
what    1
with    1
wonderful       1
yellow  2
```

使用Counter

在mapper中添加统计切词后为空的个数

```
#!/bin/env python
# encoding: utf-8

import re
import sys


seperator_pattern = re.compile(r'[^a-zA-Z0-9]+')


def print_counter(group, counter, amount):
    print >> sys.stderr, 'reporter:counter:{g},{c},{a}'.format(g=group, c=counter, a=amount)


for line in sys.stdin:
    for word in seperator_pattern.split(line):
        if word:
            print '%s\t%d' % (word.lower(), 1)
        else:
            print_counter('wc', 'empty-word', 1)
```

Streaming文档 (https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/HadoopStreaming.html#How_do_I_update_counters_in_streaming_applications)中描述打印counter的方法：

> How do I update counters in streaming applications?

> A streaming process can use the stderr to emit counter information. reporter:counter:<group>,<counter>,<amount> should be sent to stderr to update the counter.

就是向stderr中打印reporter:counter:<group>,<counter>,<amount>的字符串就可以更新counter信息了，非常简单有用的一个工具，对于job的调试和监控非常有帮助。

在集群上运行（reducer个数设置为3）

```
# 使用-files，注意：-D -files选项放在最前面，放在后面会报错，不懂为何
$ ${HADOOP_HOME}/bin/hadoop jar ${HADOOP_HOME}/share/hadoop/tools/lib/hadoop-streaming-2.6.0.jar \
    -D mapred.job.name="streaming_wordcount" \
    -D mapred.map.tasks=3 \
    -D mapred.reduce.tasks=3 \
    -D mapred.job.priority=HIGH \
    -files "mapper.py,reducer.py" \
    -input /user/<username>/wordcount/input \
    -output /user/<username>/wordcount/output \
    -mapper "python mapper.py" \
    -reducer "python reducer.py"


# output 不同的版本可能输出有所不同 -D这里使用的老配置名，前面会有一些警告，这里未显示
packageJobJar: [mapper.py, reducer.py, /tmp/hadoop-unjar707084306300214621/] [] /tmp/streamjob5287
15/09/29 10:35:14 INFO client.RMProxy: Connecting to ResourceManager at xxxxx/x.x.x.x:y
15/09/29 10:35:14 INFO client.RMProxy: Connecting to ResourceManager at xxxxx/x.x.x.x:y
15/09/29 10:35:15 INFO mapred.FileInputFormat: Total input paths to process : 3
15/09/29 10:35:15 INFO mapreduce.JobSubmitter: number of splits:3
15/09/29 10:35:15 INFO Configuration.deprecation: mapred.reduce.tasks is deprecated. Instead, use m
15/09/29 10:35:15 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1440570785607_1597
15/09/29 10:35:15 INFO impl.YarnClientImpl: Submitted application application_1440570785607_1597
15/09/29 10:35:15 INFO mapreduce.Job: The url to track the job: http://xxxxx:yyy/proxy/application_
15/09/29 10:35:15 INFO mapreduce.Job: Running job: job_1440570785607_1597
15/09/29 10:37:15 INFO mapreduce.Job: Job job_1440570785607_1597 running in uber mode : false
15/09/29 10:37:15 INFO mapreduce.Job:  map 0% reduce 0%
15/09/29 10:42:17 INFO mapreduce.Job:  map 33% reduce 0%
15/09/29 10:42:18 INFO mapreduce.Job:  map 100% reduce 0%
15/09/29 10:42:23 INFO mapreduce.Job:  map 100% reduce 100%
15/09/29 10:42:24 INFO mapreduce.Job: Job job_1440570785607_1597 completed successfully
15/09/29 10:42:24 INFO mapreduce.Job: Counters: 50
        File System Counters
                FILE: Number of bytes read=689
                FILE: Number of bytes written=661855
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=822
                HDFS: Number of bytes written=379
                HDFS: Number of read operations=18
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=6
        Job Counters
                Launched map tasks=3
                Launched reduce tasks=3
                Rack-local map tasks=3
                Total time spent by all maps in occupied slots (ms)=10657
                Total time spent by all reduces in occupied slots (ms)=21644
                Total time spent by all map tasks (ms)=10657
                Total time spent by all reduce tasks (ms)=10822
                Total vcore-seconds taken by all map tasks=10657
                Total vcore-seconds taken by all reduce tasks=10822
                Total megabyte-seconds taken by all map tasks=43651072
                Total megabyte-seconds taken by all reduce tasks=88653824
        Map-Reduce Framework
                Map input records=15
                Map output records=72
                Map output bytes=527
                Map output materialized bytes=725
                Input split bytes=423
```

```
                Combine input records=0
                Combine output records=0
                Reduce input groups=50
                Reduce shuffle bytes=725
                Reduce input records=72
                Reduce output records=50
                Spilled Records=144
                Shuffled Maps =9
                Failed Shuffles=0
                Merged Map outputs=9
                GC time elapsed (ms)=72
                CPU time spent (ms)=7870
                Physical memory (bytes) snapshot=3582062592
                Virtual memory (bytes) snapshot=29715922944
                Total committed heap usage (bytes)=10709630976
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=399
        File Output Format Counters
                Bytes Written=379
        wc
                empty-word=15
15/09/29 10:42:24 INFO streaming.StreamJob: Output directory: /user/<username>/wordcount/output
```

命令输出的需要关注的几个地方

1. The url to track the job: http://xxxxx:yyy/proxy/application_1440570785607_1597/ 点击这个url可以通过web页面查看任务的状态
2. map 0% reduce 0% 显示任务map和reduce的进度
3. 最后的Counters信息，包含系统默认的counter，可以自定义counter来统计一些任务的状态信息
4. Output directory: /user/<username>/wordcount/output 结果输出目录

常见问题和解决方法

集群Python环境的问题

使用Archive来上传一份Python的二进制环境

```
$ wget https://www.python.org/ftp/python/2.7.10/Python-2.7.10.tgz
$ tar xzf Python-2.7.10.tgz
$ cd Python-2.7.10

# compile
$ ./configure --prefix=/home/<username>/wordcount/python27

$ make -j

$ make install

# 打包一份python27.tar.gz
$ cd /home/<username>/wordcount/
$ tar czf python27.tar.gz python27/

# 上传至hadoop的hdfs
$ ${HADOOP_HOME}/bin/hadoop fs -mkdir -p /tools/
$ ${HADOOP_HOME}/bin/hadoop fs -put python27.tar.gz /tools

# 启动任务，使用刚才上传的Python版本
$ ${HADOOP_HOME}/bin/hadoop jar ${HADOOP_HOME}/share/hadoop/tools/lib/hadoop-streaming-2.6.0.jar \
    -D mapred.reduce.tasks=3 \
    -files "mapper.py,reducer.py" \
    -archives "hdfs://xxxxx:9000/tools/python27.tar.gz#py" \
    -input /user/<username>/wordcount/input \
    -output /user/<username>/wordcount/output \
    -mapper "py/python27/bin/python mapper.py" \
    -reducer "py/python27/bin/python reducer.py"
```

Reduce多路输出

有时候我们的MapReduce程序的输出希望是输出两份不同的数据，这种情况下可以使用多路输出。

旧版本使用的是outputformat，org.apache.hadoop.mapred.lib.SuffixMultipleTextOutputFormat和org.apache.hadoop.mapred.lib.SuffixMultipleSequenceFileOutputFormat是支持多路输出的，输出的格式是由原来的<key, value>变成<key, value#suffix>，suffix是A-Z，如果为其他会报错，不同suffix代表不同的输出，支持26路输出。最终的输出文件会有part-xxxx-A和part-xxxx-B等，与不同的suffix相对应。

新版本只剩下MultipleOutputs (https://hadoop.apache.org/docs/r2.6.0/api/org/apache/hadoop/mapreduce/lib/output/MultipleOutputs.html)，我暂时未找到在Streaming中使用的方法。

Map多路输入

配置多个-input的时候可以进行多路输入，在实际中可能需要对不同的输入进行不同的处理，这个时候需要获取输入的路径信息， 来区分是哪个输入路径或文件。Streaming提供了Configured_Parameters (https://hadoop.apache.org/docs/r2.6.0/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html#Configured_Parameters)， 可以获取一些运行时的信息。

Name	Type	Description
mapreduce.job.id	String	The job id
mapreduce.job.jar	String	job.jar location in job directory
mapreduce.job.local.dir	String	The job specific shared scratch space
mapreduce.task.id	String	The task id
mapreduce.task.attempt.id	String	The task attempt id
mapreduce.task.is.map	boolean	Is this a map task
mapreduce.task.partition	int	The id of the task within the job
mapreduce.map.input.file	String	The filename that the map is reading from
mapreduce.map.input.start	long	The offset of the start of the map input split
mapreduce.map.input.length	long	The number of bytes in the map input split
mapreduce.task.output.dir	String	The task’s temporary output directory

在Streaming job运行的过程中，这些mapreduce的参数格式会有所变化，所有的点（.）会变成下划线（_）。例如，mapreduce.job.id变成mapreduce_job_id。 所有的参数都可以通过环境变量来获取。

回到上面的问题，可以通过mapreduce.map.input.file来获取输入的路径名称。

```
import os

input_file = os.environ['mapreduce_map_input_file']
```

输出结果使用 Gzip 压缩

Hadoop 默认支持 Gzip 压缩，在 streaming 中只需要添加以下配置即可将输出结果压缩。

```
-D mapreduce.output.fileoutputformat.compress=true
-D mapreduce.output.fileoutputformat.compress.codec=org.apache.hadoop.io.compress.GzipCodec
```

对 Gzip 压缩数据的读取，Hadoop 是可以自行处理的，无需特殊指明输入是 Gzip 压缩。

Gzip 的特点是压缩比比较高，Hadoop 原生支持，缺点是压缩效率并不是很高，压缩比和效率不可兼得，需要考虑其他压缩方式。

压缩算法的 codec 默认也自带了多种，部分压缩算法（下面标有 native 的）需要其对应 C++ 的动态库才可以使用。

```
org.apache.hadoop.io.compress.DefaultCodec (native zlib，一般系统自带了)
org.apache.hadoop.io.compress.SnappyCodec (native snappy)
org.apache.hadoop.io.compress.GzipCodec
org.apache.hadoop.io.compress.BZip2Codec
org.apache.hadoop.io.compress.Lz4Codec (native lz4)
```

需要注意的是：压缩格式不是全部都是可以切分的，下面是找到的部分参考资料，有些说法互相有冲突，可能是不同的版本支持不一样吧，后续需要进一步查阅和学习一下，看看如何检测压缩是否为可切分的。

Choosing a Data Compression Format (http://www.cloudera.com/documentation/enterprise/5-5-x/topics/admin_data_compression_performance.html)

> For MapReduce, if you need your compressed data to be splittable, BZip2, LZO, and Snappy formats are splittable, but GZip is not.

Best splittable compression for Hadoop input = bz2? (http://stackoverflow.com/questions/14820450/best-splittable-compression-for-hadoop-input-bz2)

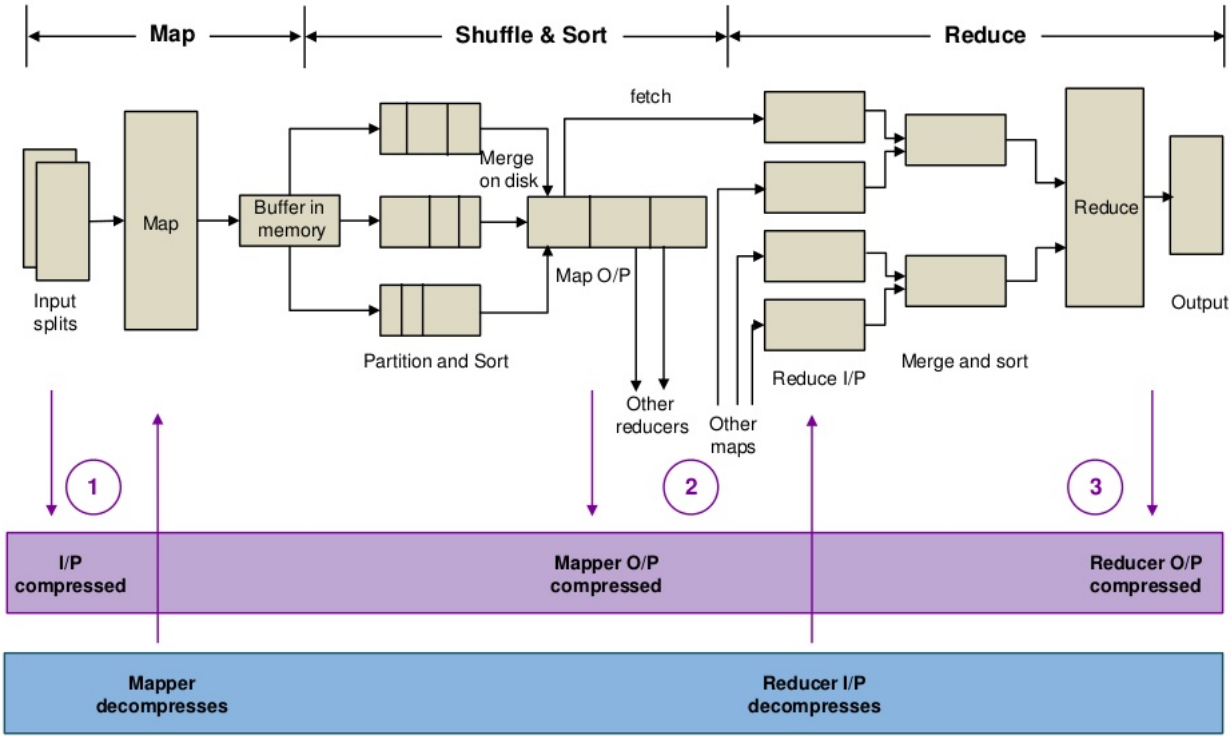
> BZIP2 is splittable in hadoop - it provides very good compression ratio but from CPU time and performances is not providing optimal results, as compression is very CPU consuming.

> LZO is splittable in hadoop - leveraging hadoop-lzo you have splittable compressed LZO files. You need to have external .lzo.index files to be able to process in parallel. The library provides all means of generating these indexes in local or distributed manner.

> LZ4 is splittable in hadoop - leveraging hadoop-4mc you have splittable compressed 4mc files. You don’t need any external indexing, and you can generate archives with provided command line tool or by Java/C code, inside/outside hadoop. 4mc makes available on hadoop LZ4 at any level of speed/compression-ratio: from fast mode reaching 500 MB/s compression speed up to high/ultra modes providing increased compression ratio, almost comparable with GZIP one.

Compression Options in Hadoop - A Tale of Tradeoffs (http://www.slideshare.net/Hadoop_Summit/singh-kamat-june27425pmroom210c)

Data Compression in Hadoop's MR Pipeline



Source: Hadoop: The Definitive Guide, Tom White

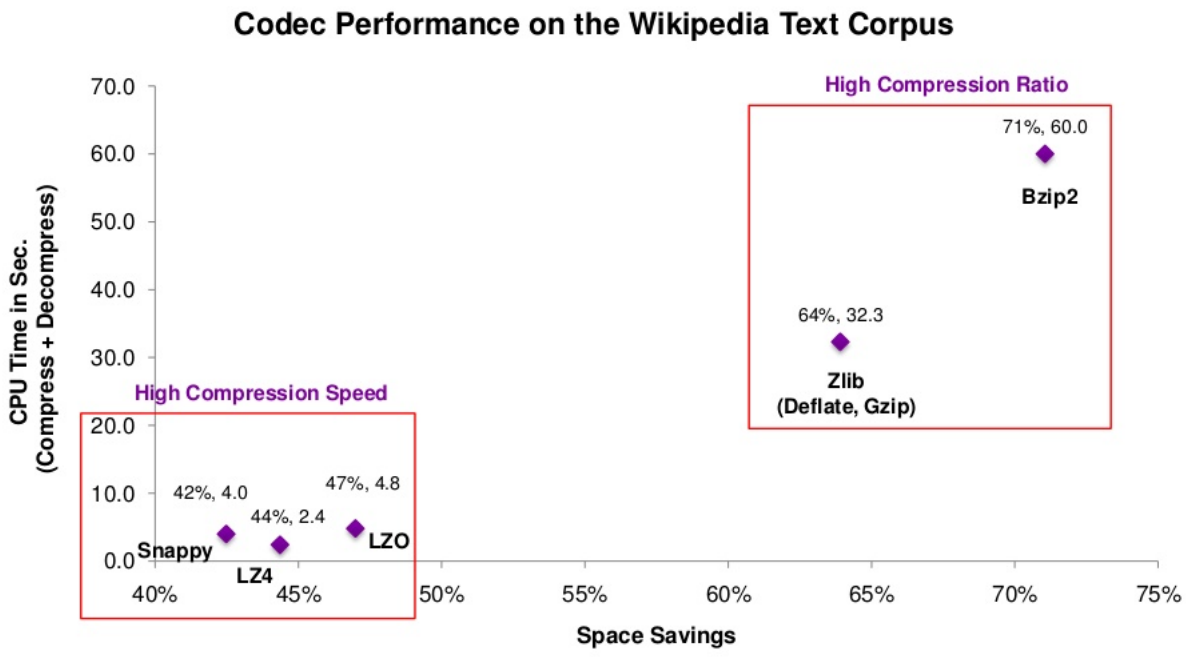
Format	Algorithm	Strategy	Emphasis	Comments
zlib	Uses DEFLATE (LZ77 and Huffman coding)	Dictionary-based, API	Compression ratio	Default codec
gzip	Wrapper around zlib	Dictionary-based, standard compression utility	Same as zlib, codec operates on and produces standard gzip files	For data interchange on and off Hadoop
bzip2	Burrows-Wheeler transform, MTF	Transform-based, block-oriented	Higher compression ratios than zlib	Common for Pig
LZO	Variant of LZ77	Dictionary-based, block-oriented, API	High compression speeds	Common for intermediate compression, HBase tables
LZ4	Simplified variant of LZ77	Fast scan, API	Very high compression speeds	Available in newer Hadoop distributions
Snappy	LZ77	Block-oriented, API	Very high compression speeds	Came out of Google, previously known as Zippy

Format	Codec (Defined in io.compression.codecs)	File Extn.	Splittable	Java/ Native
zlib/ DEFLATE (default)	org.apache.hadoop.io.compress.DefaultCodec	.deflate	N	Y/ Y
gzip	org.apache.hadoop.io.compress.GzipCodec	.gz	N	Y/ Y
bzip2	org.apache.hadoop.io.compress.BZip2Codec	.bz2	Y	Y/ Y
LZO (download separately)	com.hadoop.compression.lzo.LzoCodec	.lzo	N	N/ Y
LZ4	org.apache.hadoop.io.compress.Lz4Codec	.lz4	N	N/ Y
Snappy	org.apache.hadoop.io.compress.SnappyCodec	.snappy	N	N/ Y

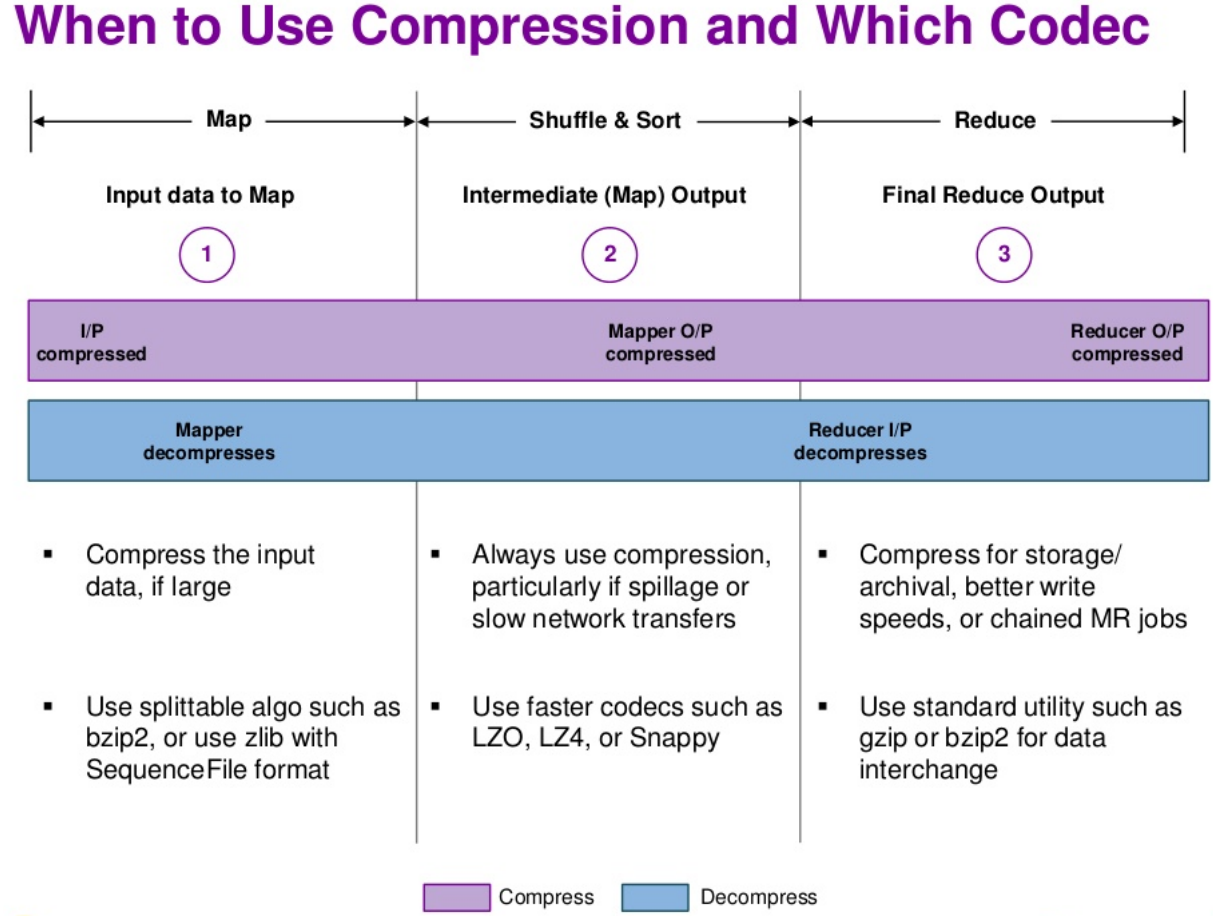
NOTES:

- **Splittability –** Bzip2 is "splittable", can be decompressed in parallel by multiple MapReduce tasks. Other algorithms require all blocks together for decompression with a single MapReduce task.
- **LZO** – Removed from Hadoop because the LZO libraries are licensed under the GNU GPL. LZO format is still supported and the codec can be downloaded separately and enabled manually.
- **Native bzip2 codec** – added by Yahoo! as part of this work in Hadoop 0.23

Space-Time Tradeoff of Compression Options



Note:
A 266 MB corpus from Wikipedia was used for the performance comparisons.
Space savings is defined as 1- (Compressed/ Uncompressed)



其他

Python对streaming的封装的类库

1. mrjob (https://github.com/Yelp/mrjob)

Hadoop周边的类库

1. snakebite (https://github.com/spotify/snakebite)：纯Python实现的HDFS客户端